Cuckoo Search

Application : Aerodynamics

```
import numpy as np

# Define the objective function: A simplified "drag function" that we aim to
minimize
def drag_function(x):
    # x[0]: curvature, x[1]: width, x[2]: slope
    # A hypothetical drag equation (for demonstration purposes)
    return x[0]**2 + 2 * x[1]**2 + 3 * x[2]**2 + 4 * x[0] * x[1] - 2 * x[1] * x[2]

# Lévy flight function using numpy for Gamma and other computations
def gamma_function(x):
    if x == 0.5:
        return np.sqrt(np.pi)  # Special case for gamma(1/2)
    elif x == 1:
        return 1  # Special case for gamma(1)
    elif x == 2:
        return 1  # Special case for gamma(2)
    else:
        return np.math.factorial(int(x) - 1) if x.is_integer() else np.inf

def levy_flight(Lambda):
    sigma = (gamma_function(1 + Lambda) * np.sin(np.pi * Lambda / 2) /
             (gamma_function((1 + Lambda) / 2) * Lambda * 2 ** ((Lambda - 1) / 2)))
** (1 / Lambda)
    u = np.random.randn() * sigma
    v = np.random.randn()
    step = u / abs(v) ** (1 / Lambda)
    return step
```

```
# Cuckoo Search Algorithm
def cuckoo_search(n, iterations, pa, lower_bound, upper_bound):
    # Initialize nests randomly
    dim = 3  # Number of design parameters
    nests = np.random.uniform(lower_bound, upper_bound, (n, dim))

    # Evaluate fitness of initial nests
    fitness = np.array([drag_function(nest) for nest in nests])
    best_nest = nests[np.argmin(fitness)]
    best_fitness = min(fitness)

    # Cuckoo Search main loop
    for _ in range(iterations):
        for i in range(n):
            # Generate a new solution by Lévy flight
            step_size = levy_flight(1.5)
            new_nest = nests[i] + step_size * np.random.uniform(-1, 1, dim)
            new_nest = np.clip(new_nest, lower_bound, upper_bound)  # Ensure within
bounds
            new_fitness = drag_function(new_nest)

            # Replace nest if the new solution is better
            if new_fitness < fitness[i]:
                nests[i] = new_nest
                fitness[i] = new_fitness

        # Abandon a fraction of the worst nests and create new ones
        for i in range(int(pa * n)):
            nests[-(i + 1)] = np.random.uniform(lower_bound, upper_bound, dim)
            fitness[-(i + 1)] = drag_function(nests[-(i + 1)])

        # Update the best nest
```

```
        if min(fitness) < best_fitness:
            best_fitness = min(fitness)
            best_nest = nests[np.argmin(fitness)]

    return best_nest, best_fitness

# Gather user input for the algorithm
print("Welcome to the Aerodynamics Optimization using Cuckoo Search!")
n = int(input("Enter the number of nests (population size): "))
iterations = int(input("Enter the number of iterations: "))
pa = float(input("Enter the probability of abandonment (between 0 and 1): "))
lower_bound = float(input("Enter the lower bound for the design parameters: "))
upper_bound = float(input("Enter the upper bound for the design parameters: "))

# Run the Cuckoo Search algorithm
best_solution, best_drag_value = cuckoo_search(n, iterations, pa, lower_bound, upper_bound)

# Display the result
print("\nOptimization Results:")
print("Best Solution (Design Parameters):", best_solution)
print("Best Drag Value:", best_drag_value)
```

Output :

```
Aerodynamics Optimization using Cuckoo Search
Enter the number of nests (population size): 10
Enter the number of iterations: 100
Enter the probability of abandonment (between 0 and 1): 0.25
Enter the lower bound for the design parameters: -10
Enter the upper bound for the design parameters: 10

Optimization Results:
Best Solution (Design Parameters): [-2.22259487 -9.7622218  -2.62606657]
Best Drag Value: -117.25613539786823
```